GUI PROGRAMMING - A REVIEW

Learning Objectives

After studying this lesson the students will be able to:

- Identify, name and state the usage of the different components of the NetBeans IDE.
- Identify and name the various methods and properties associated with the various form controls
- Create simple applications in Java using NetBeans IDE.
- Create GUI applications using the concepts of variables and control structures.

NetBeans IDE allows us to develop applications by dragging and positioning GUI components from a palette onto a container. The GUI builder automatically takes care of the correct spacing and alignment of the different components relative to each other. The JFrame acts as a container for the elements like the JLabel, JButton, JTextArea and allows direct editing of their associated properties at the design time and run time. The related methods are used in the code to develop applications for a specific task. The concept of variables and control structures are used to simplify the code of the applications.

Puzzle [3]

This group of phrases has something amazing hidden in it. Try and find out what is so unusual about this group of phrases? Is there any specific pattern in each line? *

Bar crab

Borrow or rob

Straw warts

Live evil

A man, a plan, a canal--Panama!

Delia failed.

Evil olive

Pull up if I pull up.

Step on no pets.

Ten animals I slam in a net.

Was it a bat I saw?

Was it a car or a cat I saw?

We found an interesting pattern in the puzzle and we will use this concept to develop an application in netbeans.

Introduction

Let us take a journey back in time and think as to what the world was like 15 years ago. Amazon was a large river in South America. Yahoo was a term from Gulliver's Travels. A googol was a very large number (one followed by a hundred zeroes) and to get our tickets booked we had to go to shops called 'travel agents'. In case we fell sick and went to a hospital, our records were maintained on paper. If the doctor wanted to refer to a particular patient's record, he had to instruct a assistant to hunt for information from the pile of files.

Nowadays hospitals use computers to keep the records of patients - medical history, details on what medication to give to a patient, the prescribed dosage and also personal details that can be accessed at the click of a button. The entire information is entered into a computer using a front end that accepts the different patient details. We learnt how to design valid front end forms in class XI in Java using the NetBeans IDE. This chapter will help us recapitulate all the concepts learnt in class XI.

NetBeans IDE

It can be used to create java applications very easily using the efficient GUI builder. Let us quickly recap the different components of the NetBeans IDE:

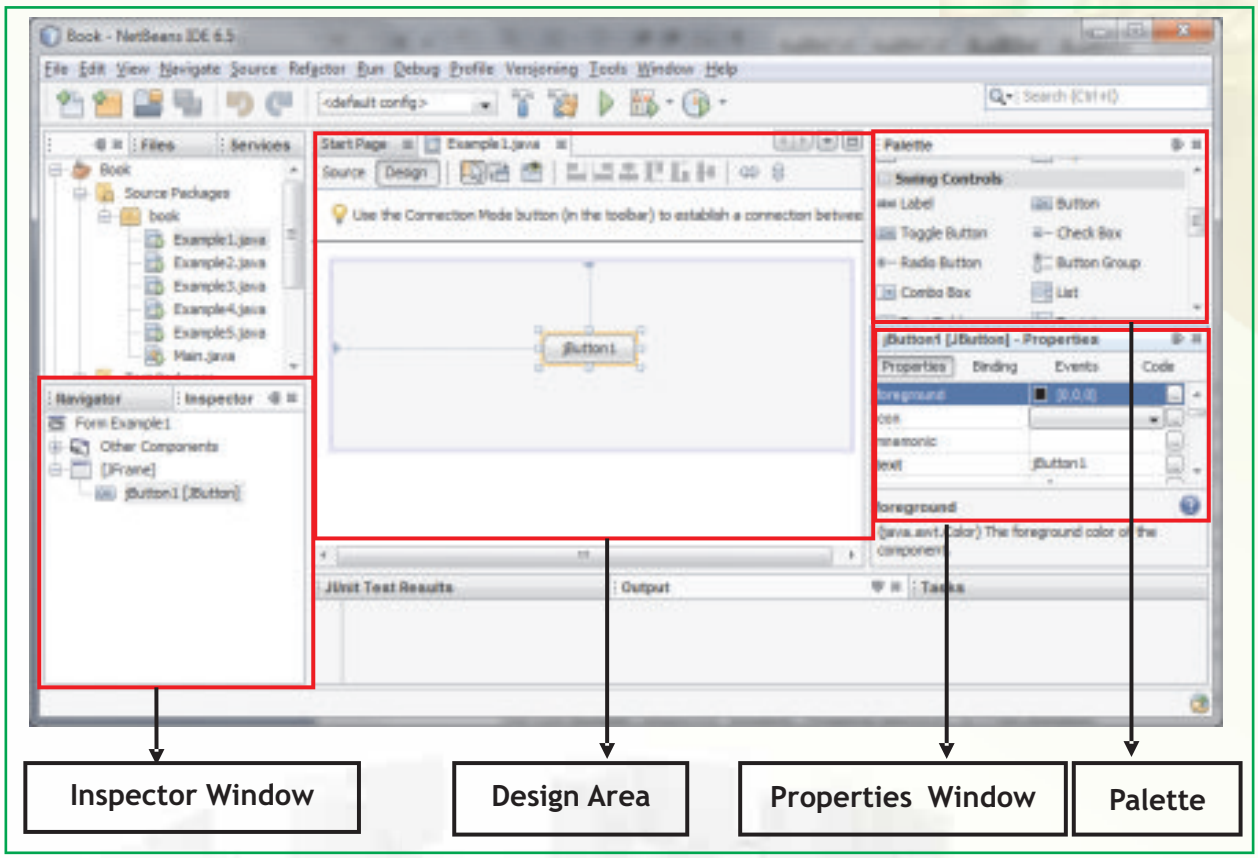

Figure 3.1 NetBeans IDE

1. Title Bar
2. Menu Bar with pull down menus
3. Toolbars
4. GUI builder: It is an area to place components on the form visually. There are two views of the GUI builder- the Design View and the Source View. We can switch over from one view to another by simply clicking on the source and design tabs directly above the Design Area.
5. Palette: Palette contains controls or components used to create GUI applications.

6. Inspector Window: This window is used to display a hierarchy of all the components or controls placed on the current form.

7. Properties Window: Using this window we can make changes in the properties of currently selected control on the form.

8. Code Editor Window: - It is the area where we write code for our java application.

Components

Components (also known as "widgets") are the basic interface elements the user interacts with: jlabels, jbuttons, jtextfields etc. Components are placed on a container (like the jFrame). There are two types of controls :

- **Parent or container controls:** They act as a background for other controls. For example-Frame. When we delete a parent control, all its child controls get deleted. When we move a parent control all its child controls also move along with it.
- **Child controls:** controls placed inside a container control are called child controls. For example-Text Field, Label, Button etc.

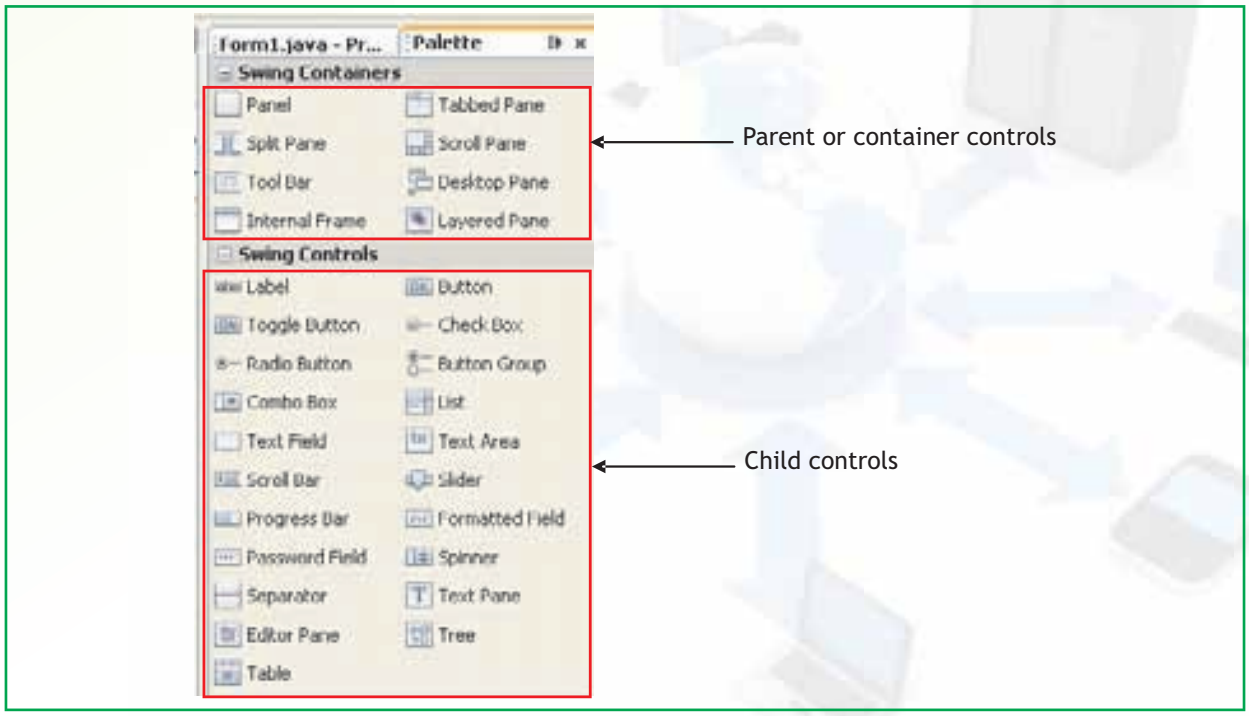

Figure 3.2 Parent and Child controls

Creating a New Project

The steps to create a new project are:

1. Select New Project from the File menu. You can also click the New Project button in the IDE toolbar.
2. In the Categories pane, select the General node. In the Projects pane, choose the Java Application type. Click the Next button.
3. Enter the name of the project in the Project Name field and specify the project location. Do not create a Main class here.
4. Click the Finish button.

Let us recap the relation between a Project, Form and Components. Each application is treated as a Project in NetBeans and each project can have one or multiple forms and this fact is clear from the Projects window as shown in Figure 3.3.

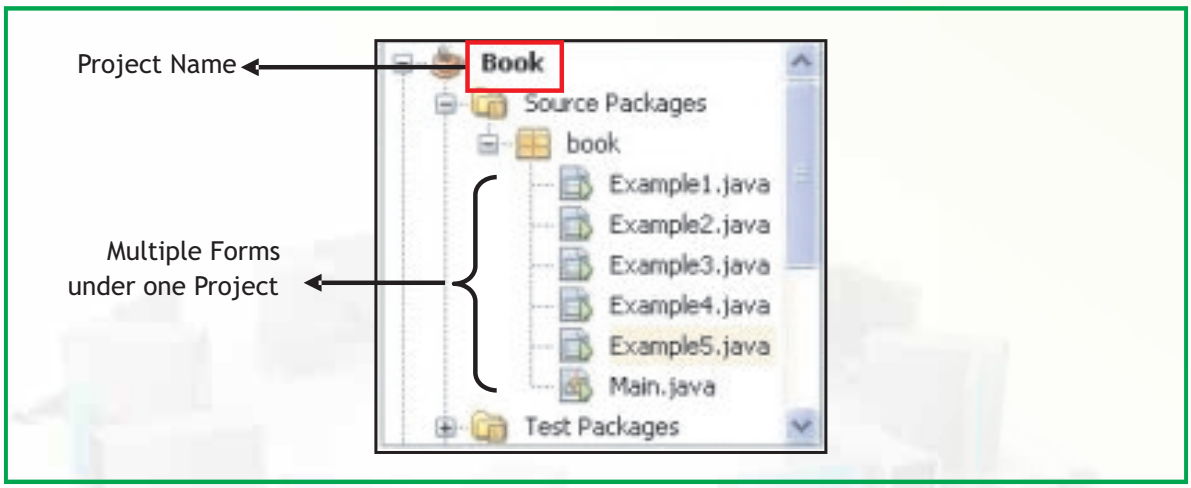

Figure 3.3 Project Window Showing Multiple Forms

Further each form can have one or more elements - some of which may be visible and some invisible. The visible components are all shown under the Frame Component and the non-visible components are part of Other components.

We use the drag and drop feature of NetBeans to place components on the form to design an effective interface for our applications. The first step that we undertook while designing our applications was adding a new jFrame form. The jFrame is a window with

title, border, (optional) menu bar and is used to contain all other components placed by the user on the form. Some of the properties of the jFrame form are defaultCloseOperation and Title.

Property	Description
defaultCloseOperation	Sets action to be performed when the user attempts to close the form.
Title	Sets the text to be displayed in the Title bar of the form window.

Figure 3.4 Properties of the jFrame Form

Any component of GUI front-end (the form itself and the swing containers and controls placed in the form) of an application is an object. Each of these objects belongs to its corresponding class predefined in Java. For example, a form is an object of JFrame class, all the textfields are objects of JTextField class, and so on. Each object has some properties, methods, and events associated with it using which you can control the object's appearance and behaviour.

Properties of an object are used to specify its appearance on the form. For example to set the **background** colour of a textfield you change its background property; to set its **font** you change its font property; and so on.

Methods are used to perform some action on the object. For example to display something in a textfield you can use its **setText()** method, to extract the contents of a textfield you can use its **getText()** method. Methods can be divided into two categories- *getters* and *setters*.

- Getters are the methods which extract some information from the object and return it to the program. Getters start with the word get. Examples of getters **are: getText(), getForeground(), getModel(), isEditable** etc.
- Setters are the methods which set some properties of the object so that the object's appearance changes. Setters start with the word set. Examples of setters are: **setText(), setForground(), setModel()** etc.

Events are the actions which are performed on controls. Examples of events are: **mouseClick, mouseMoved,keyPressed** etc. When the user performs any action on a

control, an event happens and that event invokes (sends a call to) the corresponding part of the code and the application behaves accordingly.

After setting the properties of the jFrame we can start placing components like jButton on the jFrame form. A button is a component that the user presses or pushes to trigger a specific action. When the user clicks on the button at runtime, the code associated with the click action gets executed. The various methods and properties associated with the jButton are summarized in Figure 3.5.

Property	Description
Background	Sets the background color.
Enabled	Contains enabled state of component - true if enabled else false.
Font	Sets the font.
Foreground	Sets the foreground color.
horizontal alignment	Sets the horizontal alignment of text displayed on the button.
Label	Sets the display text.
Text	Sets the display text

Method	Description
getText()	Retrieves the text typed in jButton. `String result=<button-name>.getText( );`
setEnabled	Enables or disables the button. `<button-name>.setEnabled(boolean b);`
setText()	Changes the display text at runtime. `<button-name>.setText(String text);`
setVisible	Makes the component visible or invisible - true to make the component visible; false to make it invisible. `<button-name>.setVisible(boolean aFlag);`

Figure 3.5 Properties and Methods of the jButton

We developed simple real life applications wherein on the click of the button we accepted the data from the user in the jTextField and after processing the data the result was displayed in a jTextField or a jLabel. jTextField allows editing/displaying of a single line of text. jTextField is an input area where the user can type in characters whereas a jLabel provides text instructions or information. It displays a single line of read-only text, an image or both text and image. The various methods and properties associated with the jTextField and jLabel are summarized in Figure 3.6 and 3.7 respectively.

Property	Description
Background	Sets the background color.
Border	Sets the type of border that will surround the text field.
editable	If set true user can edit textfield. Default is true.
enabled	Contains enabled state of component- True if enabled else false.
font	Sets the font.
foreground	Sets the foreground color.
horizontalAlignment	Sets the horizontal alignment of text displayed in the textField.
text	Sets the display text
toolTipText	Sets the text that will appear when cursor moves over the component.

Method	Description
getText()	Retrieves the text in typed in jTextField. `String result=<textfield-name>.getText( );`
isEditable()	Returns true if the component is editable else returns false. `boolean b=<textfield-name>.isEditable( );`

isEnabled()	Returns true if the component is enabled,else returns false. `boolean b =<textfield-name>.isEnabled( );`
setEditable	Sets whether the user can edit the text in the textField. true if editable else false. `<textfield-name>.setEditable(boolean b);`
setText()	Changes the display text at runtime. `<textfield-name>.setText(String t);`
setVisible()	Makes the component visible or invisible - true to make the component visible; false to make it invisible. `<textfield-name>.setVisible(boolean b);`

Figure 3.6 Properties and Methods of the jTextField

Property	Description
background	Sets the background color.
enabled	Contains enabled state of component- true if enabled else false.
font	Sets the font.
foreground	Sets the foreground color.
horizontalAlignment	Sets the horizontal alignment of text displayed in the component.
text	Sets the display text

Method	Description
getText()	Retrieves the text in typed in jLabel. `String result=<label-name>.getText();`
isEnabled()	Returns true if the component is enabled,else returns false. `boolean b=<label-name>.isEnabled();`
setText()	Changes the display text at runtime. `<label-name>.setText(String t);`
setVisible()	Makes the component visible or invisible - true to make the component visible; false to make it invisible. `<label-name>.setVisible(boolean b);`

Figure 3.7 Properties and Methods of the jLabel

When we had become familiar with the usage of jTextField and jLabel controls then we developed an application in which we wanted to accept multiline input and display multiline output. Well can you recall the name of the component. Exactly the component is Text Area component. This component allows us to accept multiline input from the user or display multiple lines of information. This component automatically adds vertical or horizontal scroll bars as and when required during run time. The various methods and properties associated with the jTextArea are summarized in Figure 3.8.

Property	Description
background	Sets the background color.
columns	Sets number of columns preferred for display.
editable	If set true user can edit textfield. Default is true.
enabled	Contains enabled state of component- true if enabled else false.
font	Sets the font.

foreground	Sets the foreground color.
lineWrap	Indicates whether line of text should wrap in case it exceeds allocated width. (Default is false)
rows	Sets number of rows preferred for display.
text	Sets the display text
wrapStyleWord	Sends word to next line in case lineWrap is true and it results in breaking of a word, when lines are wrapped.

Method	Description
append()	Adds data at the end. `<textarea-name>.append(String str);`
getText()	Retrieves the text in typed in jTextArea. `String str = <textarea-name>.getText();`
isEditable()	Returns true if the component is editable else returns false. `boolean b = <textarea-name>.isEditable();`
isEnabled()	Returns true if the component is enabled, else returns false. `boolean b = <textarea-name>.isEnabled();`
setText()	Changes the display text at runtime. `<textarea-name>.setText(String t);`

Figure 3.8 Properties and Methods of the jTextArea

Let us try and recollect the name of the component which can be used to enter confidential input like passwords which are single line. That's right the component is jPassword. We can suppress the display of input as this component allows us to input confidential information like passwords. Each character entered can be replaced by an echo character. By default, the echo character is the asterisk, *. The properties of jPassword are summarized below:

Property	Description
background	Sets the background color.
font	Sets the font.
foreground	Sets the foreground color.
text	Sets the display text
echoChar	Sets the character that will be displayed instead of text.

Figure 3.9 Properties of jPassword

Well we used radio buttons when we wanted to provide the user several choices and allowed him to select one of the choices (the radio buttons belong to a group allowing the user to select single option). But radio buttons occupy a lot of space. Thus, in case of too many options we used Combo boxes as they help save space and are less cumbersome to design as compared to radio button. We used check box and list when we wanted to display multiple options like selecting favourite sports or ordering multiple food items in a restaurant. The list is a preferred option over check box in situations wherever multiple options are required to be selected from a large number of known set of options as they help save space and are less cumbersome to design as compared to check boxes. The properties and methods of jRadioButton are summarized below:

Property	Description
background	Sets the background color.
buttonGroup	Specifies the name of the group of button to which the jRadioButton belongs.
enabled	Contains enabled state of component -true if enabled else false.
font	Sets the font.
foreground	Sets the foreground color.
label	Sets the display text.
text	Sets the display text.
Selected	Sets the button as selected, if set to true, default is false.

Method	Description
getText()	Retrieves the text displayed by radio button. `String str = <radiobutton-name>.getText();`
isSelected()	Returns true if the component is checked else returns false. `boolean b = <radiobutton-name>.isSelected();`
setText()	Changes the display text at runtime. `<radiobutton-name>.setText(String t);`
setSelected()	Checks(true) or unchecks the radio button. `<radiobutton-name>.setSelected(boolean b);`

Figure 3.10 Properties and methods of the jRadioButton

jCheckBox is a small box like component that is either marked or unmarked. When you click on it, it changes from checked to unchecked or vice versa automatically. The properties and methods of jCheckBox are summarized below:

Property	Description
background	Sets the background color.
buttonGroup	Specifies the name of the group of button to which the jCheckBox belongs.
font	Sets the font.
foreground	Sets the foreground color.
label	Sets the display text.
text	Sets the display text
selected	Sets the check box as selected if set to true, default is false.

Method	Description
getText()	Retrieves the text typed in `String str = <checkbox-name>.getText();`
isSelected()	Returns true if the component is checked else returns false. `boolean b = <checkbox-name>.isSelected();`
setText()	Changes the display text at runtime. `<checkbox-name>.setText(String t);`
setSelected()	Checks(true) or unchecks the checkbox. `<checkbox-name>.setSelected(boolean b);`

Figure 3.11 Properties and methods of the jCheckBox

jComboBox is like a drop down box - you can click a drop-down arrow and select an option from a list whereas jList provides a scrollable set of items from which one or more may be selected. The properties and methods of jComboBox and jList are summarized below:

Property	Description
background	Sets the background color.
buttongroup	Specifies the name of the group of button to which the jComboBox belongs.
editable	If set true user can edit ComboBox. Default is true.
enabled	Contains enabled state of component- True if enabled else false.
font	Sets the font.
foreground	Sets the foreground color.
model	Contains the values to be displayed in the combobox.
text	Sets the display text
selectedIndex	Sets the index number of the element which should be selected by default.
selectedItem	Sets the selected item in the combobox. selectedItem and selectedIndex are in synchronization with each other.

Method	Description

getSelectedItem()	Retrieves the selected item.
	`Object result =`
	`<combobox-name>.getSelectedItem();`

getSelectedIndex()	Retrieves the index of the selected item.
	`int result =`
	`<combobox-name>.getSelectedIndex();`

setModel()	Sets the data model that the combo box uses to get its list of elements.
	`<combobox-name>.setModel`
	`(ComboBoxModel aModel);`

Figure 3.12 Properties and methods of the jComboBox

Property	Description
background	Sets the background color.
enabled	Contains enabled state of component- true if enabled else false.
font	Sets the font.
foreground	Sets the foreground color.
model	Contains the values to be displayed in the list.
selectedIndex	Contains the index value of selected option of the control.
selectionMode	Describes the mode for selecting values. - SINGLE (List box allows single selection only) - SINGLE_INTERVAL (List box allows single continuous selection of options using shift key of keyboard) - MULTIPLE_INTERVAL (List box allows multiple selections of options using ctrl key of keyboard)

Method	Description
getSelectedValue()	Returns the selected value when only a single item is selected, if multiple items are selected then returns first selected value. Returns null in case no item selected `Object result= <list-name>.getSelectedValue();`
isSelectedIndex()	Returns true if specified index is selected. `boolean b = <list-name>.isSelectedIndex(int index);`

Figure 3.13 Properties and methods of the jList

We used JOptionPane when we wanted to request information from the user, display information to the user or a combination of both. It required an import statement at the top of the program. Well can you recollect the import statement? That's right it is:

```
import javax.swing.JOptionPane;
```

OR

```
import javax.swing.*;
```

Either of them is acceptable. The difference is that the latter will import the entire library as denoted by the star whereas the first statement will just import the JOptionPane library.

Method	Description
showMessageDialog()	Shows a one-button, modal dialog box that gives the user some information. Example : `JOptionPane.showMessageDialog(this,"Java and NetBeans");`

showConfirmDialog()	Shows a three-button modal dialog that asks the user a question. User can respond by pressing any of the suitable buttons.
	
	Example:
	`Confirm=`
	`JOptionPane.showConfirmDialog(null,"quit?")`

showInputDialog()	Shows a modal dialog that prompts the user for input. It prompts the user with a text box in which the user can enter the relevant input.
	
	Example :
	`name=`
	`JOptionPane.showInputDialog(this,"Name:");`

Figure 3.14 Properties and methods of the JOptionPane

Variables

Let us try and recollect why the need for variables arose. Well, we used variables when we required containers to store the values for some input, intermediate result or the final result of an operation. The characteristics of a variable are:

- It has a name.
- It is capable of storing values.
- It provides temporary storage.
- It is capable of changing its value during program execution.

Variables help us to hold value for some input coming from the user or to hold intermediate result of some calculation or the final result of an operation. In other words, variables are like containers that can be used to store whatever values are needed for a specific computation. However, as different materials require different containers, and so we used different data types to hold different values.

Java programming language requires that all variables must first be declared before they can be used.

When programming, we store the variables in our computer's memory, but the computer has to know what kind of data we want to store in them, since it is not going to occupy the same amount of memory to store a simple number or to store a single letter or a large number, and they are not going to be interpreted the same way so variables were used along with datatypes. The data types supported by java are summarized as follows:

Data type states the way the values of that type are stored, the operations that can be done on that type, and the range for that type.

Numeric Data Types :

These data types are used to store integer values only i.e. whole numbers only. The storage size and range is listed below :

Name	Size	Range	Example
byte	1 byte(8 bits)	-128 to 127(-2^7 to +(2^7-1))	byte rollno;
short	2 bytes(16 bits)	-32768 to 32767(-2^{15} to +(2^{15}-1))	short rate;
int	4 bytes(32 bits)	-2^{31} to +(2^{31}-1)	int num1;
long	8 bytes (64 bits)	-2^{63} to +(2^{63}-1)	long amount;

Figure 3.15 Storage size and range of numeric data types

Floating Data Types:

These data types are used to store numbers having decimal points i.e. they can store numbers having fractional values.

Name	Description	Size	Range	Example
float	Single precision floating point	4 bytes (32 bits)	($3.4x10^{-38}$) to +($3.4x10^{38}$)	float average;
double	Double precision floating point	8 bytes (64 bits)	($1.8x10^{-308}$) to +($1.8x10^{308}$)	double principal;

Figure 3.16 Storage size and range of floating data types

The decision about which numeric data type to use should be based on the range of values that a variable can take.

Character Data Types:

These data types are used to store characters. Character data types can store any type of values - numbers, characters and special characters. When we want to store a single character, we use char data type and when we want to store a group of characters we use string data type. For example to store grades (A, B, C, D, E) of a student we will use char type but to store name of a student, we will use string type. The char data type value is always enclosed inside '' (single quotes), whereas a string data type value is enclosed in "" (double quotes).

Operators are symbols that manipulate, combine or compare variables.

Operators

With the introduction of variables and constants there arose a need to perform certain operations on them. We performed operations on variables and constants using operators. The operators available in java are summarized below:

Assignment Operator :

One of the most common operator is the assignment operator "=" which is used to assign a value to a variable. We assign the value given on the right hand side to the variable specified on the left hand side. The value on the right hand side can be a number or an arithmetic expression. For example:

```
int sum = 0;
int prime = 4*5;
```

Arithmetic Operators :

These operators perform addition, subtraction, multiplication, and division. These symbols are similar to mathematical symbols. The only symbol that is different is "%", which divides one operand by another and returns the remainder as its result.

```
+ additive operator
- subtraction operator
```

```
* multiplication operator
/ division operator
% remainder operator
```

Relational Operator :

A relational operator is used to test for some kind of relation between two entities. A mathematical expression created using a relational operator forms a relational expression or a condition. The following table lists the various relational operators and their usage:

Operator	Meaning	Usage
==	equal to	Tests whether two values are equal.
!=	not equal to	Tests whether two values are unequal.
>	greater than	Tests if the value of the left expression is greater than that of the right.
<	less than	Tests if the value of the left expression is less than that of the right.
>=	greater than or equal to	Tests if the value of the left expression is greater than or equal to that of the right.
<=	less than or equal to	Tests if the value of the left expression is less than or equal to that of the right.

Figure 3.17 Relational Operators

Logical Operator :

A logical operator denotes a logical operation. Logical operators and relational operators are used together to form a complex condition. Logical operators are:

Operator	Use	Meaning
&&	a>10 && b<8	a and b are both true
||	a>10 || b<8	Either a or b is true
!	! a	a is false

Figure 3.18 Logical Operators

Unary Operators :

The unary operators perform different kind of operations on a single operand .The operations performed are increasing/decreasing a value, negating a value/ expression, or inverting a boolean value.

Symbol	Name of the Operator	Operation	Example
+	Unary plus operator	indicates positive value	`num = +1;`
-	Unary minus operator	negates an expression	`num = - num;`
++	Increment operator	increments a value by 1	`num = ++ num;`
--	Decrement operator	decrements a value by 1	`num = -- num;`

Figure 3.19 Unary Operators

Increment/Decrement Operators :

The increment/decrement (++,--) operators can be a prefix or a postfix. In a pre increment/decrement expression (++ x or -- x), an operator is applied before an operand while in a post increment/decrement expression (x++ or x --) an operator is applied after an operand. In both conditions 1 is added to the value of the variable and the result is stored back to the variable. However, in a prefix expression, value is incremented first then this new value is restored back to the variable. In postfix expression the current value is assigned to a variable then it is incremented by 1 and restored back to the original variable.

Let us now try and recollect the conversion methods that we have used in java. When a Java program receives input data from a user, it must often convert it from one form (e.g., String) into another (e.g., double or int) for processing.

Conversion methods

- To convert a string value to a number (For example, to convert the String value in a text field(jTextField1) to an int, long, float or double), we can use parse methods. Assume the following declarations:

```
String num1=jTextField1.getText();
int sum; long product;
float amount;  double simple_int;
```

Type	Example statement
int	sum=Integer.parseInt(num1);
int	sum=Integer.parseInt(jTextField1.getText());
long	product=Long.parseLong(num1);
long	product=Long.parseLong(jTextField1.getText());
float	amount=Float.parseFloat(num1);
float	amount=Float.parseFloat(jTextField1.getText());

double	simple_int=Double.parseDouble(num1);
	OR
	simple_int=Double.parseDouble(jTextField1.getText());

Figure 3.20 Usage of Parse Methods

- To convert a number to string we used **valueOf** method. Assume the following code:

```
int i    = 100;
float f  = (float) 200.0;
double d = 400.0;
long l   = 100000;
string str1 = String.valueOf(i);
string str2 = String.valueOf(f);
string str3 = String.valueOf(d);
string str4 = String.valueOf(l);
jTextField1.setText("Values of int, float, double and long
are "+str1+", "+str2+", "+str3+" and "+str4);
```

Figure 3.21 Usage of valueOf Method

- To convert a number to string, we also used **toString** method. Assume the following code :

```
int i    = 100;
float f  = (float) 200.0;
double d = 400.0;
long l   = 100000;
string str1 = Integer.toString(i);
string str2 = Float.toString(f);
string str3 = Double.toString(d);
string str4 = Long.toString(l);
jTextField1.setText("Values of int, float, double and long
are "+str1+", "+str2+", "+str3+" and "+str4);
```

Figure 3.22 Usage of toString Method

- To convert a number to string, we also used concatenation operator(+). If either operand of a concatenation is a string, the other operand is converted to string. Assume the following code :

```
int sum = 103;
String convertsum;
convertsum = "" + sum;
    // Converts int 103 to String "103"
convertsum = sum + " is a number";
    // Assigns "103 is a number" to convertsum.
convertsum = "" + 10.9;
    // Assigns "10.9" to convertsum
convertsum = "" + 10.0/3.0;
    // Assigns "3.3333333333333333" to convertsum
```

Figure 3.23 Concatenation operator

Control Structures

We used control structures when we wanted to control the flow of the program. We learnt two types of control structures in class XI namely, Selection statements and Iteration statements.

Control structures allow us to control the flow of our program's execution. If left unchecked by control-flow statements, a program's logic will flow through statements from top to bottom. We can have some control on the flow of a program by using operators to regulate precedence of operations, but control structures provide the power to change statement order and govern the flow of control in a program.

Selection Statements:

A selection statement selects among a set of statements depending on the value of a controlling expression. The selection statements are the if statement and the switch statement, which are discussed below:

Simple if Statement - The if statement allows selection (decision making) depending upon the outcome of a condition. If the condition evaluates to true then the statement immediately following if will be executed and otherwise if the condition evaluates to false then the statements following the else clause will be executed. The selection statements are also called conditional statements or decision statements.

The syntax of if statement is as shown below:

Syntax:

```
if (conditional expression)
{
    Statement Block;
}
else
{
    Statement Block;
}
```

Points to remember about if statement :

- The conditional expression is always enclosed in parenthesis.
- The conditional expression may be a simple expression or a compound expression.
- Each statement block may have a single or multiple statements to be executed. In case there is a single statement to be executed then it is not mandatory to enclose it in curly braces ({}) but if there are multiple statements then they must be enclosed in curly braces ({}).
- The else clause is optional and needs to be included only when some action is to be taken if the test condition evaluates to false.

Nested if . . . else - These control structures are used to test for multiple conditions as against the simple if statement which can be used to test a single condition. The syntax of nested if else is as follows:

Syntax:

```
if (conditional expression1)
{
    statements1;
}
else if (conditional expression2)
{
    statements2;
}
else if (conditional expression3)
{
    statements3;
}
else
{
    statements4;
}
```

Now let us design a form as shown in figure 3.24

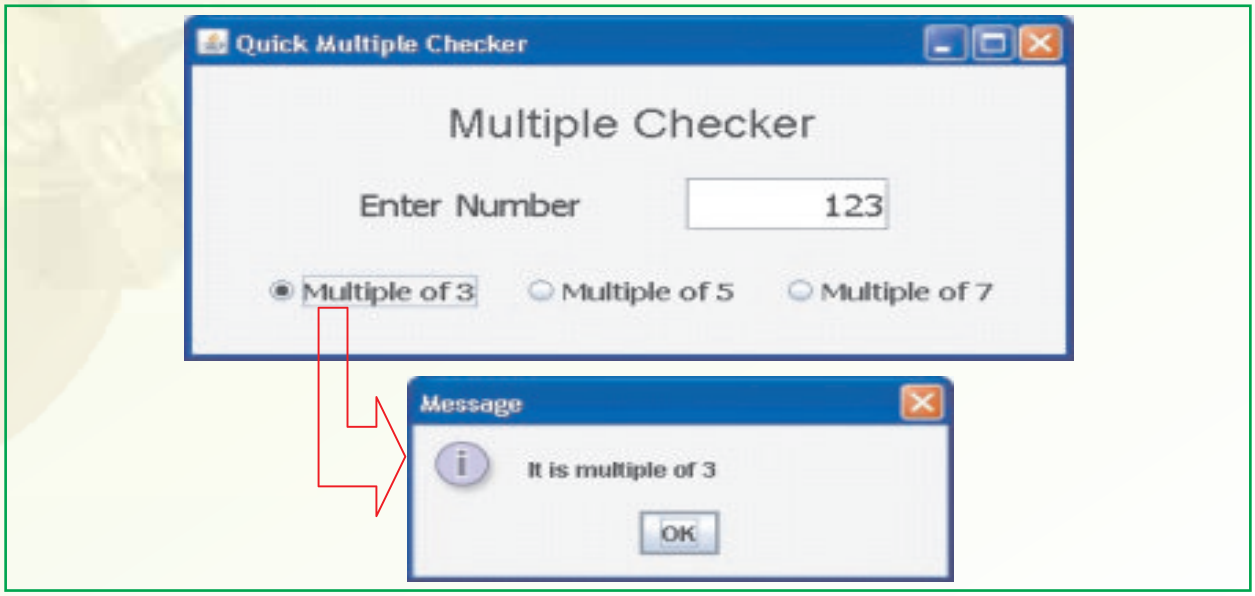

Figure 3.24 To check whether the given number is a multiple of 3,5 or 7

Follow the steps enumerated below to design the form:

1. Add a new JFrame Form and change its title property to Quick Multiple Checker.
2. Add a label set its Text as Enter Number
3. Add a jTextField and set its initial Text as ""
4. Add three radio buttons on the form - Set the text of each of them as "Multiple of 3", "Multiple of 5" and "Multiple of 7".Group the radio buttons so that only one can be selected.

In this application we ask the user to enter a number and then the user will select one of the radio buttons and depending upon the button selected the multiple check for that number will be performed. Let us now write the code for the above mentioned application. Code to check for multiple of 3 is given. Try and write the code to perform a similar check for multiple of 5 and multiple of 7.

```
private void
jRadioButton1ActionPerformed(java.awt.event.ActionEvent evt)
{
  double Number=Double.parseDouble(jTextField1.getText());
  if (Number%3==0)
   JOptionPane.showMessageDialog(this,"It is multiple of 3");
 else
   JOptionPane.showMessageDialog
                (this,"It is NOT multiple of 3");
}
```

Figure 3.25 Code for Multiple Checker

Let us now understand this code in detail.

`Double.parseDouble(jTextField1.getText())`

- retrieves the value entered by the user in the text field using getText().This value by default is treated as a string and not as a number so it needs to be converted to a double type and this is achieved using the parseDouble() method.The value is then stored in the variable Number.

`if (Number%3=0)`

- check whether the number when divided by 3 gives 0 as the remainder.If the remainder is zero then the number is divisible by 3 else it is not.

Switch Statement - This selection statement allows us to test the value of an expression with a series of character or integer values. On finding a matching value the control jumps to the statement pertaining to that value and the statement is executed, till the break statement is encountered or the end of switch is reached. The expression must either evaluate to an integer value or a character value. It cannot be a string or a real number. The syntax of the switch statement is as follows:

```
switch (Variable/Expression)
{
    case Value1:statements1 ;
                break ;
    case Value2:statements2 ;
                break ;
                .
                .
    default:statements3 ;
}
```

Comparing Switch and If..else Statements - Switch is used to select sections of code depending on specific integer or character values. If we are handling specific coded values (eg, the number of the button that was clicked in a JOptionPane), or processing characters(whose codes are treated like numbers), then switch is useful. The limitations of switch are as follows:

- It doesn't allow ranges, eg case 90-100.
- It requires either integers or characters and doesn't allow useful types like String.

```
String comment; // The generated result.
int choice = Integer.ParseInt(jTextField.getText);
                //Enter 0, 1, or 2.
switch (choice)
{
    case 0: comment = "You look so much better than usual.";
            break;
    case 1: comment = "Your work is up to its usual standards.";
            break;
```

```
        case 2: comment =
            "You're quite competent for so little experience.";
                break;
        default: comment =
            "Oops -- something is wrong with this code.";
    }
```

Equivalent if statement

```
String comment; // The generated result.
int choice= Integer.ParseInt(jTextField.getText);
          //Enter is 0, 1,or 2.
if (choice == 0)
     comment = "You look so much better than usual.";
else if (choice == 1)
     comment = "Your work is up to its usual standards.";
else if (choice == 2)
  comment="You're quite competent for so little experience";
else
  comment = "Oops -- something is wrong with this code.";
```

A switch statement can often be rewritten as an if statement. Let us look at the example given above, when a selection is to be made based on a single value, the switch statement is generally easier to read. The switch is useful when you need to manage a lot of **if /else if / else**. It has a shorter syntax and is more appropriate in this case.

Let us now design an application in which we will calculate the selling price depending upon the profit percent selected by the user. Design the application as shown in figure 3.26. Set the relevant properties of the components.

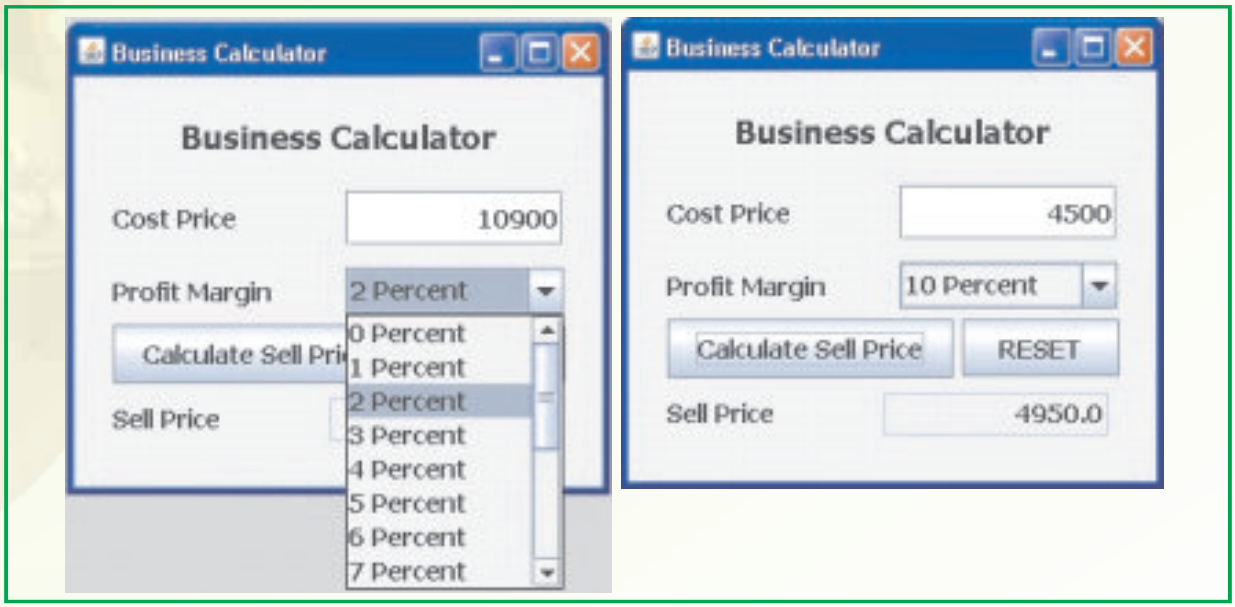

Figure 3.26 To calculate selling price depending upon profit margin

In this application depending upon the profit percent selected by the user we will calculate the selling price .Let us now write the code.

```
//Business Calculator

private void
jButton1ActionPerformed(java.awt.event.ActionEvent evt)

{

    double CP,Profit=0,SP;

    CP=Double.parseDouble(jTextField1.getText());

    Profit=jComboBox1.getSelectedIndex();

    SP=CP+CP*(Profit/100);

    jTextField2.setText(Double.toString(SP));

}
```

```
private void
jButton2ActionPerformed(java.awt.event.ActionEvent evt)
{
    jTextField1.setText("");
    jComboBox1.setSelectedIndex(0);
    jTextField2.setText("");
}
```

Figure 3.27 Code to calculate selling price

Let us now understand the code in detail.

`Double.parseDouble(jTextField1.getText())`

- retrieves the value entered by the user in the text field using getText().This value by default is treated as a string and not as a number so it needs to be converted to a double type and this is achieved using the parseDouble() method. The value is then stored in the variable CP.

`jComboBox.getSelectedIndex()`

- retrieves the index of the selected item so if the user selects 1st item profit is 0 and if the user selects 3rd, item profit is 2.This value is saved in a variable profit.

`SP=CP+CP*(Profit/100)`

- Formula to calculate selling Price.

`jTextField2.setText(Double.toString(SP)`

- The variable SP is a numeric value so it is converted to a string using the toString() method and then the value is displayed in the text field using the setText() method.

In the above application the profit margin is obtained from the Index value of the selected item of the combo box but what happens if the values in the combobox are as shown in the figure 3.28.

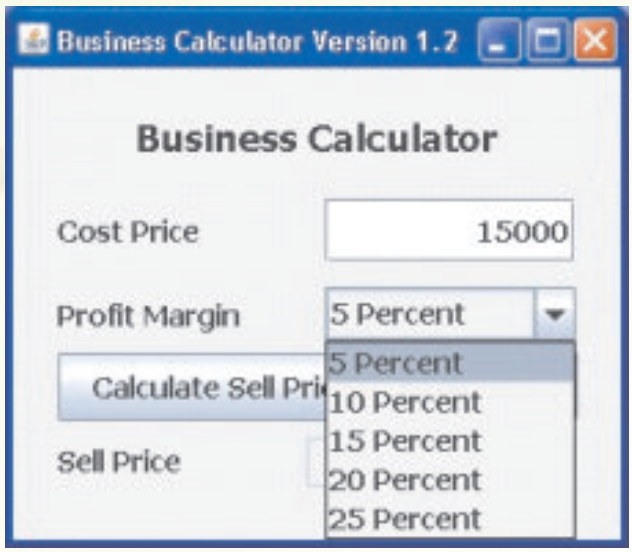

Figure 3.28 Business Calculator version 1.2

We will use switch case to write the code. Only the code using switch case is given below but it is recommended that you think and try writing the code on your own.

```
private void
jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{
    double CP,Profit=0,SP;
    CP=Double.parseDouble(jTextField1.getText());
    switch (jComboBox1.getSelectedIndex())
    {
   case 0:Profit=5;break;//if first item selected profit is 5
   case 1:Profit=10;break;//if second item selected profit is 10
   case 2:Profit=15;break;
   case 3:Profit=20;break;
   case 4:Profit=25;break;
   default:Profit=0;break;
}
```

Figure 3.29 Code for Business Calculator version 1.2

Iteration Statements :

These statements are used to perform a set of instructions repeatedly while the condition is true. Iteration statements are also called looping statements.

for loop - The loop has four different elements that have different purposes. These elements are:

a) **Initialization expression:** Before entering in a loop, its variables must be initialized.

b) **Test Expression:** The test expression decides whether the loop body will be executed or not. If the test condition is true, the loop body gets executed otherwise the loop is terminated.

c) **Increment/Decrement Expression:** The Increment/Decrement expression changes the value of the loop variable.

d) **The Body of the loop:** The statements, which are executed repeatedly while the test expression evaluates to true form the body of the loop.

The syntax of the for loop is:

Syntax

```
for( initialization; test exp; increment/decrement exp)
{
statements;
}
```

The three expressions inside the round braces of for loop are optional. Using this fact an infinite loop can be created as follows:

```
for (int I = 1 ;I <= 5 ;I++ )
{
    jTextArea1.setText (jTextArea1.getText()+I);
    //It will display 12345 in jTextArea1
}
```

While Loop - The while loop is an entry-controlled loop. It means that the loop condition is tested before executing the loop body. If the loop condition is initially false, for the first iteration, then loop may not execute even once. The main characteristic of the while loop is that it can be used in both cases i.e. when the number of iterations is known as well as when it is unknown. The syntax of the while loop is as follows:

Syntax

```
while(test expression)
{
    loop body
}
```

Remember that in while loop, a loop control variable should be initialized before the loop begins and the loop variable should be updated inside the body of the while loop (else it will become an endless loop).

Do..While Loop - Do..While loop is an exit-controlled loop. In the do..while loop, the test occurs at the end of the loop. This ensures that the do..while loop executes the statements included in the loop body at least once. After the first execution of the statement, it evaluates the test expression. If the expression evaluates to true, then it executes the statements of the loop body again. Like if and while statements, the condition being checked must be included between parenthesis. The while statement must end with a semicolon. The syntax of the loop is as follows:

Syntax

```
do
{
    loop body
}
while (test expression);
```

Comparing Do...While and While - The difference between do-while and while is that do-while evaluates its expression at the end of the loop instead of at the beginning. Therefore, the statements within the do block are always executed at least once. Do-while is an exit controlled loop and while is an entry controlled loop.

```
int ctr;
ctr=0;
while (ctr < 5)    // Test and Loop
{
   JOptionPane.showMessageDialog(this,ctr);
                   // Display Number
   ctr = ctr + 1;
                   // Increment LoopCounter
}
```

The same code is written using do while. In do while counter is checked at the end of loop body, therefore loop body is executed at least once.

```
int ctr = 0;
do                 // Loop Begins
{
    JOptionPane.showMessageDialog(this,ctr);
                   // Display Number
    ctr = ctr + 1;
                   // Increment LoopCounter
}
while (ctr<5);     // Testing the value of counter
```

Let us now develop a few applications which will help us to strengthen our programming concepts learnt in class XI.

Let us now design an application in which we will generate the series and its sum. Design the application as shown in figure 3.30. Set the relevant properties of the components.

Series and its Sum

1+2+3+4

2+4+6+8+10

2+7+12+17+22

Figure 3.30 Sum of Series Calculator

Let us now write the code as follows:

```
private void
jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{
    int Start,Step,Terms,Sum=0;
    Start=Integer.parseInt(jTextField1.getText());
                    //Start point of series
```

```
        Step =jComboBox1.getSelectedIndex()+1;
                        //step in the series
        Terms=Integer.parseInt(jTextField2.getText());
                        // no of terms
        for (int I=1;I<=Terms;I++)
        {
            Sum+=Start;
            Start+=Step;
        }
        jTextField3.setText(Integer.toString(Sum));
    }
    private void
    jButton2ActionPerformed(java.awt.event.ActionEvent evt)
    {
        jTextField1.setText("");
        jTextField2.setText("");
        jTextField3.setText("");
                //Index -1 indicates no selection
        jComboBox1.setSelectedIndex(-1);
    }
```

Figure 3.31 Code to generate the series and the Sum of Series

Let us now understand the code in detail.

`Integer.parseInt(jTextField1.getText())`

- retrieves the value entered by the user in the text field using getText().This value by default is treated as a string and not as a number so it needs to be converted to an integer type and this is achieved using the parseInt() method. The value is then stored in the variable Start.

```
jComboBox.getSelectedIndex()+1
```

- retrieves the index of the selected item so if the user selects 1st item as Step then 1 is added to the index value of the 1st item. This value is saved in a variable Step.

```
Integer.parseInt(jTextField1.getText())
```

- retrieves the value entered by the user in the text field using getText().This value by default is treated as a string and not as a number so it needs to be converted to an integer type and this is achieved using the parseInt() method.The value is then stored in the variable Terms.

Now we have the value from where the user wants to start the series, the terms in the series and the step between numbers of the series.

```
for (int I=1;I<=Terms;I++)
{
        Sum+=Start;
        Start+=Step;
}
jTextField3.setText(Integer.toString(Sum));
```

- the for loop is executed as many times as the Terms and the Step is added to Start which in turn is added to Sum and finally the Sum of the numbers is displayed.

```
jTextField1.setText("");
jTextField2.setText("");
jTextField3.setText("");
jComboBox1.setSelectedIndex(-1);
```

- On clicking the reset button all the three textfields are set to "" and the setselectedIndex value of combobox is set to -1. Index -1 indicates that no item is selected.

Let us now develop an application to reverse the given number and also find out whether the number is a palindrome or not.(The pattern that we found in the phrases was that each line had a palindrome hidden in it.)

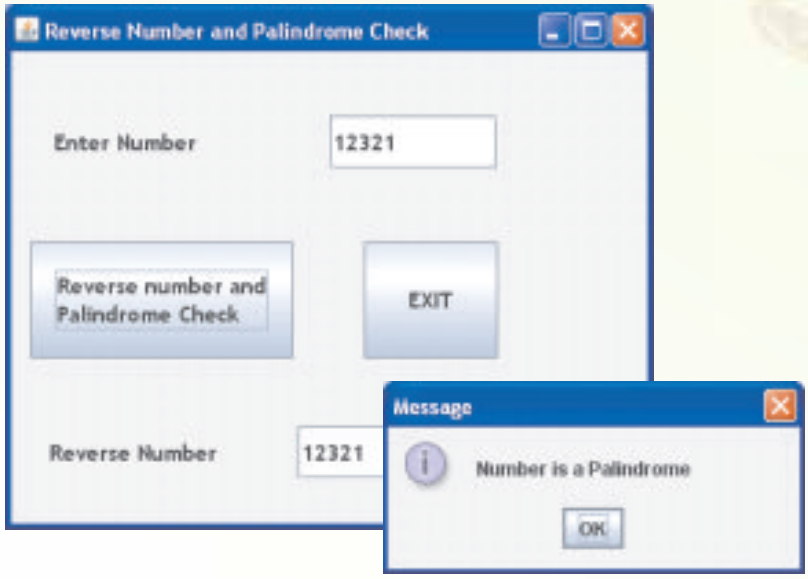

Figure 3.32 Reverse Number and Palindrome Check

Let us now write the code for reversing a given number and check whether the given number is a palindrome or not.

```
 //Reverse Number Program with Palindrome Check
private void
jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{
    long Number,Temp,RevNumber=0;
    Number=Long.parseLong(jTextField1.getText());
    Temp=Number;
```

```
        while (Temp>0) // till temp > 0 continue to perform the loop
        {
            RevNumber=(RevNumber*10)+(Temp%10);
            // RevNumber is multiplied by 10 and added to the
            // remainder of temp divided by 100
            Temp=Temp/10;
        }
        jTextField2.setText(Long.toString(RevNumber));
        if (Number==RevNumber)
            JOptionPane.showMessageDialog
                            (this,"Number is Palindrome");
        else
            JOptionPane.showMessageDialog
                            (this,"Number is not a Palindrome");
    }
    private void
    jButton2ActionPerformed(java.awt.event.ActionEvent evt)
    {
        jTextField1.setText("");
        jTextField2.setText("");
    }
```

Figure 3.33 Code for Reverse Number and Palindrome Check

Let us now understand the code in detail :

```
Long.parseLong(jTextField1.getText())
```

- retrieves the value entered by the user in the text field using getText().This value by default is treated as a string and not as a number so it needs to be converted to long type and this is achieved using the parsetLog() method. The value is then stored in the variable Number.

```
Temp=Number
```

- the number is stored in another variable temp.

```
while (Temp>0)
{
    RevNumber=(RevNumber*10)+(Temp%10);
    Temp=Temp/10;
}
```

- while loop will continue as long as temp is greater than zero. In this loop we divide temp by 10 and obtain the remainder. The remainder is then added to the value in the variable RevNumber multiplied by 10.Initially RevNumber=0.The value of variable temp is divided by 10 every time the loop is executed. The loop is executed as long as temp is greater than 0.

```
if (Number==RevNumber)
    JOptionPane.showMessageDialog
                        (this,"Number is Palindrome");
else
    JOptionPane.showMessageDialog
                        (this,"Number is not Palindrome");
```

- If the reverse of the number, which is stored in variable RevNumber and the number entered by the user which is stored in Number are the same then the number is a palindrome otherwise it is not a palindrome.

We have revised the concepts learnt in class XI. Now in the next few chapters we will enhance our knowledge of NetBeans and delve further into programming concepts.

Summary

- NetBeans is an IDE using which we can develop GUI applications in Java.
- NetBeans provides various components used to create a GUI front-end interface.
- GUI components' appearance and behaviour is controlled by their properties and methods.
- We should use meaningful names for controls on the form and variables in the code. It makes programming convenient.
- Some useful Data Types supported in Java are: int, double, char and boolean.
- String is an Object (reference) type supported in Java.
- A variable must be declared before it can be used.
- Different types of operators are available in Java. Operators are used to perform various operations on data.
- Control Statements available in java are: if..else, switch..case, for, while, do..while.

EXERCISES

MULTIPLE CHOICE QUESTIONS

1. What will be the output of the program given below. Which number is printed twice?

```
int sum1 = 3;
sum1++;
jTextField1.setText(""+sum1);
++sum1;
jTextField2.setText(""+sum1);
jTextField3.setText(""+(++sum1));
```

```
jTextField4.setText(""+sum1++);
jTextField5.setText(""+sum1);
```

a. 5 b. 6

c. 4 d. 7

2. Consider the following code snippet :

```
int anumber=14;
    if (anumber >=10){
    if (anumber == 10)
    jLabel1.setText("first string");
    else jLabel1.setText("second string");
    jLabel2.setText("third string");
```

What will be the output when anumber=14

a. first string b. second string

c. first string d. second string
third string third string

3. What value for X above will print EXACTLY 20 lines to standard output :

```
int count = 0;
while(count++ < X ) {
jTextField1.setText("Line " + count);
}
```

a. 0 b. 5

c. 10 d. 11

4. What will be the value of total after the loop finishes execution.

```
int total = 0; // must be initialized before the for (int count =
5; count <=10; count++ )
{
total += count;
}
jTextField1.setText("The total is " + total);
```

a. 10 b. 16

c. 45 d. 36

5. What's wrong with the while statement?

```
while( (ctr < 5) && (ctr > 30))
```

a the logical operator && cannot be used in a test condition.

b the while loop is an exit-condition loop.

c the test condition is always false.

d the test condition is always true.

6. If there is more than one statement in the block of a for loop, which of the following must be placed at the beginning and the ending of the loop block?

a parentheses () b French curly braces { }

c brackets [ ] d arrows < >

7. Given the following information:

int a = 11;

int b = 22;

int c = 33;

int d = 11;

Which of the following statements are true :

i) a = = b ii) b != d iii) c <= b

iv) a < c v) a = = d vi) c > a

vii) a >= c

a i),iv) & vii) b ii),iv), v) & vi)

c ii),iv), vi) & vii) d iii),v),vi) & vii)

8. The statement i++; is equivalent to

a i = i + i; b i = i + 1;

c i = i - 1; d i - - ;

ANSWER THE FOLLOWING QUESTIONS

1. Explain the following terms:
   a) IDE
   b) Inspector Window
   c) Form
2. Differentiate between :
   a) TextField and TextArea
   b) ComboBox and ListBox
   c) getText() and setText()
3. What is the significance of the following properties in TextArea ?
   LineWrap WrapStyleWord
4. What are list type controls used for ?
5. How would you determine whether a combo box is editable or not ?
6. List different selection modes of a list.
7. What is a button group? Which control is generally used with a buttongroup.
8. Write and explain two methods each of check box and radio button.

LAB EXERCISES

1. Design a GUI application in which the user enters a number in the text field and on clicking the button the sum of the digits of the number should be displayed in a label.

   Hint : Suppose user enters 123 the output should be 6(1+2+3).

2. Design a GUI application to accept a String from the user in a text field and print using option pane whether it is a palindrome or not.

   Hint ABBA is a palindrome.

3. Design a GUI application to accept the cost price and selling price form the user in two text fields then calculate the profit or loss incurred.

4. Design a GUI application to accept a character in a text field and print in a label if that character is a vowel: a, e, i, o, or u. The application should be case sensitive.

5. Design a GUI application that repeatedly accepts numbers in a option pane and once the typed number is 0 the maximum and minimum of all numbers typed are displayed.

6. Design a GUI application in java to convert temperature from Celsius to Fahrenheit or vice versa using radio buttons and two text fields

7. Design a GUI application in java to convert kilograms into grams, litres into milliliters, rupees into paisa using combobox and text fields.

8. A book publishing house decided to go in for computerization. The database will be maintained at the back end but you have to design the front end for the company. You have to accept book code, Title, Author and Quantity sold from the user. The Price will be generated depending upon the book code. Net price should be calculated on the basis of the discount given.

   Bookseller - 25%

   School - 20%

   Customer - 5%

9. A networking company decided to computerize its employee salary . Develop an application to store employee's personal data which will be saved in the back end. The front end should accept Name, Father's Name, Mother's Name, Address, Gender, Basic Salary, Medical and Conveyance. Calculate gross and net salary.

Basic	DA	HRA
>=40000	35%	37%
>=20000	25%	32%
>=10000	25%	30%

TEAM BASED TIME BOUND EXERCISE

(Team size recommended : 3 students in each team)

1. Students will visualize the details that have to be provided while creating a member registration for an e-reservation site. Each team has to design a layout for the form. The form has to be designed using NetBeans IDE. The team has to specify the following:
   - Controls that will be used to develop the application.
   - Data types of variables to be used.
   - Validations to be performed while accepting the data.
2. Students will visualize the details that have to be provided while developing a domicile certificate for a student of class XII. Each team has to design the form using NetBeans IDE. The team has to specify the following:
   - Controls that will be used to develop the application
   - Data types of variables to be used.
   - Validations to be performed while accepting the data.